

Name	
Student Id	
Signature	

**The University of New South Wales**  
**November 2003**  
**Final Examination**

COMP3421/COMP9415  
Computer Graphics

- Time allowed: three hours
- Number of Questions: 4
- Answer **All** Questions
- Total number of marks: 100
- Calculators permitted.
- You may keep this paper.
- Answer each question in a separate booklet.
- Write your answers using ink.

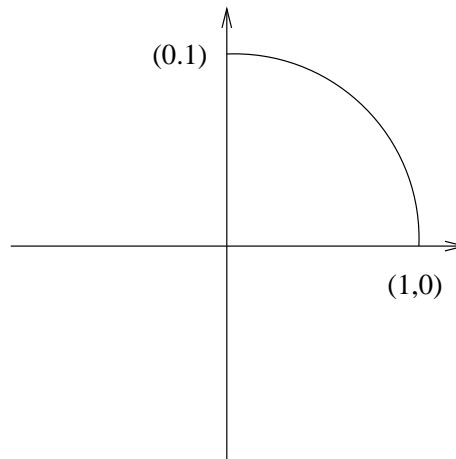
### Question 1 (20 Marks)

Give brief (no more than two sentences) definitions of the following

- (a) secondary ray
- (b) bump map
- (c) VRAM
- (d) surface of revolution
- (e) fractal

### Question 2 (25 Marks)

- (a) We have a wall that has four corners at  $(0,0,0)$ ,  $(1,0,0)$ ,  $(1,1,0)$ ,  $(0,1,0)$ . That is, it is one unit high and one unit long, starts at the origin and is parallel to the x axis. Find the transformation matrix that turns it into the wall with corners  $(1,0,0)$ ,  $(1,0,2)$ ,  $(1,2,2)$ ,  $(1,2,0)$ . That is, the new wall is two units high and two units long, starts at  $(1,0,0)$  and is parallel to the x axis.
- (b) We have a grey triangle with  $k_d = 0.5$  illuminated with a single distant light with intensity 0.6 so that the direction to the light is  $(0, 3, 4)$ . The triangle has corners  $(0,0,0)$ ,  $(1,0,0)$  and  $(0,1,0)$  and is flat shaded. There is no ambient or specular reflection. What intensity will the triangle be shaded?
- (c) We want a Bezier curve that approximates one quarter of the circle with centre  $(0,0)$  and radius 1. The curve should extend from  $(1,0)$  to  $(0,1)$ .



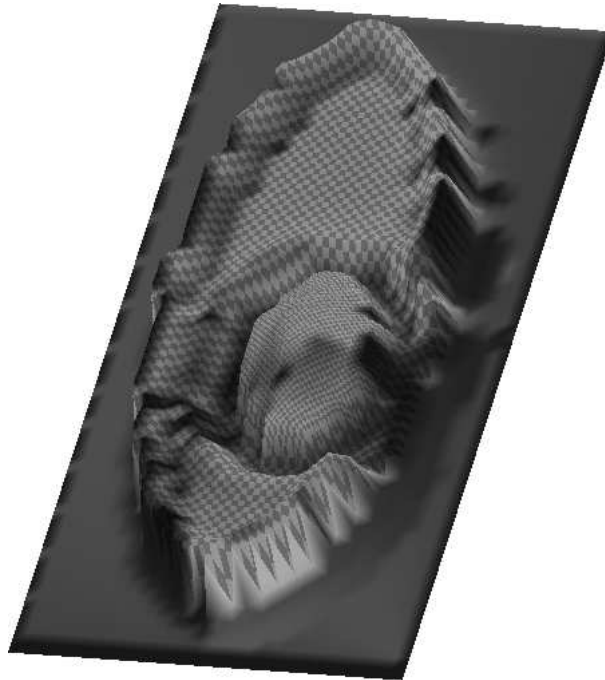
First work out the midpoint ( $t = 1/2$ ) of the Bezier curve with control points  $p_0, p_1, p_2, p_3$ .

Next, find the four control points of a Bezier curve that best approximates the quarter circle. (Hint The curve should start at  $(1, 0)$  with a vertical tangent, its midpoint should be the midpoint of the quarter circle at  $(\sqrt{2}/2, \sqrt{2}/2)$ , and end at  $(0, 1)$  with a horizontal tangent.)

### Question 3 (25 Marks)

A researcher has a simulation of the cells in the human heart that trigger the heartbeat. These cells work by creating regular pulses of electricity that propagate to neighbouring cells, causing them to create pulses of electricity that propagate further. The simulation calculates the strength of the electric field in each of a thousand cells at a thousand points in time. Each cell has a x and y coordinates. There are several different types of cell in the simulation.

The figure below gives an idea of the sort of thing the researcher would like to see. It uses the strength of the electric field at one particular time to set the Z coordinate of each cell and different textures to show different types of cells. She would like to be able to animate this to show the field at different times.



This isn't the only way she would like to see the data, she would also like to see things like how rapidly the field is changing, how long it has been over a particular threshold and so on.

Design a system to help this researcher visualise her data.

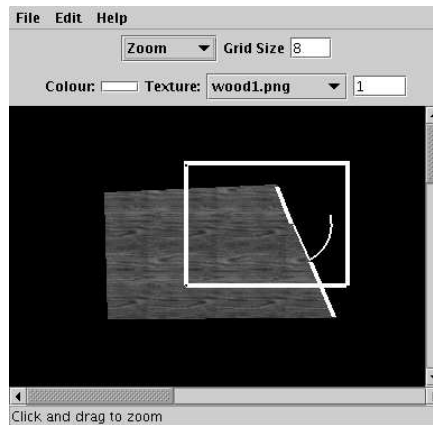
Tell me

- A: what hardware your system would require,
- B: what software and algorithms your system would use,
- C: how you would store the information needed,
- D: and how a user would interact with your system.

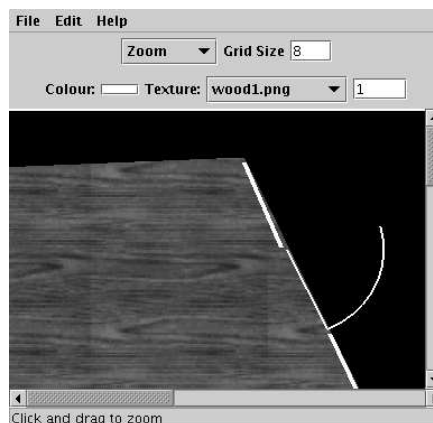
#### Question 4 (30 Marks)

In this question you will modify my solution to the first assignment to add a zoom tool that allows users to zoom in and get a close view of a part of the floor plan.

Here is an example of my solution in action. After selecting the zoom tool I drag out a rectangle with the left mouse:



When I release the mouse button, the contents of the rectangle expand to fill the drawing area.



The way this is implemented is by setting the window (as in window-to-viewport transformation) using the OpenGL method

```
gluOrtho2D( minx, maxx, miny, maxy);
```

which sets the window with the specified values.

At the end of the paper you will find a listing of part of my solution to the first assignment. When you want to change one of my files you can indicate it by saying something like “Delete line 63 of FPEDCanvas.java and replace with the following code.”

One small change has been made to my solution. Once you have zoomed in, it is possible to create `FPEDShapes` with co-ordinates that are not integers, so methods that used to take a `Point` (int co-ordinates) now take a `Point2D` (double co-ordinates). For example,

```
public abstract void addPoint(Point p);
```

was changed to

```
public abstract void addPoint(Point2D p);
```

Each of the parts of this question can be done independantly of each other. That is, you can assume part a works when you write part b, even if you haven't worked out how to do part a.

- (a) Write a `FPEDRectangle` class that is a subclass of `FPEDShape`. This will be used by the Zoom tool to draw the rectangle interactively. A `FPEDRectangle` is defined by two control points at opposite corners of the rectangle. Hint: This is easy if you extend the right class.

Your `FPEDRectangle` must also have four methods:

```
public double xmin();
public double xmax();
public double ymin();
public double ymax();
```

that return the minimum and maximum x and y coordinates of the `FPEDRectangle`.

- (b) Modify the `FPEDCanvas` class so that it uses a `FPEDRectangle` to define the window (as in the window-to-viewport transformation). In particular, it must have methods:

```
public void setWindow(FPEDRectangle window);
public FPEDRectangle getTheWindow();
```

`setWindow` should define the window used in `gluOrtho2D` to set the window part of the the window to viewport transformation. Getting this to work properly with the scroll bars that pan the window is a little tricky, so you don't need to do it—you can just ignore the scroll bars.

- (c) Now write `ZoomTool`. Dragging a with the left mouse should cause a rectangle to be interactively drawn. When the mouse is released, the window is set to that rectangle.
- (d) To simplify your work in the assignment, I set things up so that the window-to-viewport transformation was just a translation—that is, screen and world co-ordinates were the same except that `(xoffset,yoffset)` was subtracted from the world coordinates to get screen coordinates. Once you've zoomed in or out, the window-to-viewport transformation is not just a translation, and screen co-ordinates (type `Point` returned in a `MouseEvent`) can't be converted to world coordinates be just adding an offset.

Modify `Tool` so that it makes the appropriate conversion between co-ordinate systems. Hint: The viewport has `(0,0)` as upper left corner, and `FPEDCanvas.width` and `FPEDCanvas.height` are its width and height respectively.

## FPEDCanvas.java

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.*;
4  import java.io.*;
5  import net.java.games.jogl.*;
6  import javax.swing.*;
7  import javax.swing.event.*;
8
9  /** This class represents the drawing area in the floor plan editor
10     @author Tim Lambert
11  */
12
13  public class FPEDCanvas extends JPanel
14     implements GLEventListener{
15
16
17     /** list of the shapes displayed in this canvas */
18     protected FPEDShapeList shapes;
19     /** the editor window that contains this canvas */
20     protected FPEDEdit editor;
21     /** currently selected shapes */
22     protected FPEDShape selection = null;
23
24
```

```

25     /** OpenGL drawing canvas */
26     GLCanvas glc = null;
27
28     JScrollBar vertical;
29     JScrollBar horizontal;
30
31     //width and height on screen
32     protected int width=600, height=400;
33
34     //maximum width and height
35     protected int maxWidth = 1000, maxHeight = 1000;
36
37     /** The constructor is given the name of the file containing the
38         floor plan to be displayed in this canvas and the parent frame.
39         @param name - the name of the file containing the image map
40         @param editor - the parent window
41     */
42     public FPEDCanvas(String name, FPEDEdit editor) {
43         this.editor = editor;
44         shapes = new FPEDShapeList();
45         selection = null;
46         if (name.equals("untitled.fpe")) { // new file
47         } else {
48             try {
49                 shapes.read(name);
50             } catch (IOException e) {
51                 JOptionPane.showMessageDialog(editor,
52                     "Sorry, couldn't open file\n"+e,
53                     "Open error",
54                     JOptionPane.ERROR_MESSAGE);
55             } catch (MalformedShapeException e) {
56                 JOptionPane.showMessageDialog
57                     (editor,
58                     "Are you sure that file contains a floor plan?\n"+e,
59                     "Read error",
60                     JOptionPane.ERROR_MESSAGE);
61             }
62         }
63
64     /**
65      * Set the layout manager for the Frame. This is BorderLayout
66      * which will cause the children ( the GLDrawable ) to
67      * be resized to fill the Frame upon resizing the Frame.
68      */
69     setLayout( new BorderLayout() );
70
71     /**
72      * Create a new GLDrawable from the factory
73      */
74     GLCapabilities caps = new GLCapabilities();
75
76     glc =
77         GLDrawableFactory.getFactory().createGLCanvas(caps);
78
79     /** Add the GLDrawable to the Panel and display it */
80     add( "Center", glc );
81     setSize(width,height);
82
83     /**
84      * Add the GLEventListener to handle GL events
85      * and initialize!
86      */
87     glc.addGLEventListener( this );
88
89     /** vertical scroll bar */
90     vertical = new JScrollBar();
91     vertical.setMaximum(maxHeight);
92     add("East", vertical);
93     vertical.getModel().addChangeListener(new ChangeListener() {
94         public void stateChanged(ChangeEvent e){
95             glc.repaint();

```

```

96         }
97     }
98         );
99
100     /* horizontal scroll bar */
101     horizontal = new JScrollBar(JScrollBar.HORIZONTAL);
102     horizontal.setMaximum(maxWidth);
103     add("South", horizontal);
104     horizontal.getModel().addChangeListener(new ChangeListener() {
105         public void stateChanged(ChangeEvent e){
106             glc.repaint();
107         }
108     }
109         );
110 }
111
112 public int getXoffset() {
113     return horizontal.getValue();
114 }
115
116 public int getYoffset() {
117     return vertical.getValue();
118 }
119
120
121 /** pass add/remove*Listener calls onto the GLDrawable */
122 public void addMouseMotionListener(MouseMotionListener l) {
123     glc.addMouseMotionListener( l );
124 }
125 public void addMouseListener(MouseListener l) {
126     glc.addMouseListener( l );
127 }
128 public void addKeyListener(KeyListener l) {
129     glc.addKeyListener( l );
130 }
131 public void removeMouseMotionListener(MouseMotionListener l) {
132     glc.removeMouseMotionListener( l );
133 }
134 public void removeMouseListener(MouseListener l) {
135     glc.removeMouseListener( l );
136 }
137 public void removeKeyListener(KeyListener l) {
138     glc.removeKeyListener( l );
139 }
140 /** pass repaint to the GLDrawable */
141
142 public void repaint() {
143     if (glc != null) glc.repaint();
144 }
145
146 public void paintComponent(Graphics g) {
147     super.paintComponent(g);
148     glc.repaint();
149 }
150
151 /** This method writes the content of the canvas to a file
152     @param filename - the name of the file
153     */
154
155 public void write(String filename) {
156     shapes.write(filename);
157 }
158
159 /** Return the shapes on this canvas.
160     @returns list of shapes on this canvas
161     */
162 public FPEDShapeList getShapes(){
163     return shapes;
164 }
165
166 /** Select a shape (as needed by the Select tool

```

```

167     @param r - shape to select
168     */
169     public void select(FPEDShape r){
170         selection = r;
171     }
172
173
174     /** Return the selected shape
175     @returns selected shapes
176     */
177     public FPEDShape getSelection(){
178         return selection;
179     }
180
181
182     /**
183     * Executed exactly once to initialize the
184     * associated GLDrawable
185     */
186
187     public void init( GLDrawable drawable ) {
188         /**
189         * Set the background colour when the GLDrawable
190         * is cleared
191         */
192         GL gl = drawable.getGL();
193         gl.glClearColor( 0.0f, 0.0f, 0.0f, 1.0f ); //black
194
195     }
196
197     /** Handles resizing of the GLDrawable */
198
199     public void reshape( GLDrawable drawable, int x, int y,
200         int width, int height ) {
201         GL gl = drawable.getGL();
202         GLU glu = drawable.getGLU();
203         gl.glViewport( x, y, width, height );
204         this.width = width;
205         horizontal.setVisibleAmount(width);
206         this.height = height;
207         vertical.setVisibleAmount(height);
208     }
209
210     /** This method handles the painting of the GLDrawable */
211
212     public void display( GLDrawable drawable ) {
213
214         GL gl = drawable.getGL();
215         GLU glu = drawable.getGLU();
216         gl.glMatrixMode( GL.GL_PROJECTION );
217         gl.glLoadIdentity();
218         int xoffset = horizontal.getValue();
219         int yoffset = vertical.getValue();
220         glu.gluOrtho2D( xoffset, xoffset+width, yoffset+height, yoffset);
221         /** Clear the colour buffer */
222         gl.glClear( GL.GL_COLOR_BUFFER_BIT );
223         shapes.paint(gl,glu, drawable);
224         if (selection != null){
225             selection.paintSelect(gl,glu);
226         }
227     }
228     /** This method handles things if display depth changes */
229     public void displayChanged(GLDrawable drawable,
230         boolean modeChanged,
231         boolean deviceChanged){
232     }
233
234 }

```

## FPEDPolygon.java

```
1  /** This class represents a polygon */
2  import java.awt.*;
3  import java.util.*;
4  import net.java.games.jogl.*;
5
6
7  public class FPEDPolygon extends FPEDShape {
8
9      protected ArrayList pts2d;
10     protected int selection = -1;
11
12     public FPEDPolygon() {
13         pts2d = new ArrayList();
14     }
15
16
17     /** paint this curve into g.*/
18     public void paint(GL gl, GLU glu, GLDrawable glc){
19
20         if (fill != null || texture != null){
21             if (texture == null) {
22                 setColor(gl,fill);
23                 gl.glDisable( GL.GL_TEXTURE_2D );
24             } else {
25                 gl.glEnable( GL.GL_TEXTURE_2D );
26                 gl.glTexEnvf( GL.GL_TEXTURE_ENV, GL.GL_TEXTURE_ENV_MODE,
27                             GL.GL_REPLACE );
28                 gl.glBindTexture( GL.GL_TEXTURE_2D, texture.name(gl, glu, glc) );
29             }
30             gl.glBegin( GL.GL_POLYGON ); //draw the polygon
31             for(int i = 0; i < pts2d.size(); i++) {
32                 Point2D p = (Point2D)(pts2d.get(i));
33                 gl.glTexCoord2d(scale*p.x/100, -scale*p.y/100); //dodgy texture scaling
34                 gl.glVertex2d(p.x, p.y);
35             }
36             gl.glEnd();
37
38         }
39     }
40
41     public void paintSelect(GL gl, GLU glu){
42         for(int i = 0; i < pts2d.size(); i++){
43             Point2D p = (Point2D)(pts2d.get(i));
44             drawPoint(gl,p.x,p.y);
45         }
46     }
47
48     static final int EPSILON = 6; /* distance for picking */
49
50     /** return index of control point near to (x,y) or -1 if nothing near */
51     public Point selectPoint(Point2D p) {
52         double mind = Double.MAX_VALUE;
53         selection = -1;
54         for (int i = 0; i < pts2d.size(); i++) {
55             Point2D p2 = (Point2D)(pts2d.get(i));
56             double d = sqr(p2.x-p.x) + sqr(p2.y-p.y);
57             if (d < mind && d < EPSILON*EPSILON) {
58                 mind = d;
59                 selection = i;
60             }
61         }
62
63         if (selection == -1) {
64             return null;
65         } else {
66             return ((Point2D)(pts2d.get(selection))).toPoint();
67         }
68     }
69
70     // square of a double
```

```

71     static double sqr(double x) {
72         return x*x;
73     }
74
75     /** add a control point */
76     public void addPoint(Point2D p) {
77         pts2d.add(p);
78         selection = pts2d.size() - 1;
79     }
80
81     /** find the closest edge of a polygon to p and select end point
82     and return poition on that edge as number between 0 and 1*/
83     protected double closestEdge(Polygon pl, Point2D p) {
84         Point2D closest = null;
85         int closesti;
86         for (int i = 0; i < pl.npoints; i++){
87             int iplus = (i+1) % pl.npoints;
88             Edge2D e = new Edge2D(pl.xpoints[i],pl.ypoints[i],pl.xpoints[iplus],pl.ypoints[iplus]);
89             Point2D pe = e.toLineSpace(p);
90             if (pe.getX() > 0 && pe.getX() < 1 && Math.abs(pe.getY()) < EPSILON) {
91                 if (closest == null ||
92                     Math.abs(pe.getY()) < Math.abs(closest.getY()) ) {
93                     closest = pe;
94                     selection = iplus;
95                 }
96             }
97         }
98         return (closest == null) ? -1.0 : closest.getX();
99     }
100 }
101
102 /** set selected control point */
103 public void setPoint(Point2D p) {
104     if (selection >= 0) {
105         pts2d.set(selection, p);
106     }
107 }
108
109 /** remove specified control point */
110 public void removePoint() {
111     if (selection >= 0) {
112         pts2d.remove(selection);
113     }
114     selection = -1; //otherwise next control point becomes selected
115 }
116
117 public boolean contains(Point2D p){
118     Polygon pts = new Polygon();
119     for (int i = 0; i < pts2d.size(); i++){
120         Point pi = ((Point2D)(pts2d.get(i))).toPoint();
121         pts.addPoint(pi.x,pi.y);
122     }
123     Point pp = p.toPoint();
124     return pts.contains(pp.x,pp.y);
125 }
126
127 public void rotate(Point2D fixed, double angle){
128     apply(Matrix2D.rotateAbout(fixed, angle));
129 }
130
131 public void translate(double deltax, double deltax){
132     apply(new Matrix2D(1,0,deltax,0,1,deltax));
133 }
134
135 public Point2D centre(){
136     Point2D p = ((Point2D)(pts2d.get(0)));
137     double minx, miny, maxx, maxy;
138     minx = maxx = p.x;
139     miny = maxy = p.y;
140     for (int i = 1; i < pts2d.size(); i++){

```

```

142         p = ((Point2D)(pts2d.get(i)));
143         if (p.x < minx){
144             minx = p.x;
145         }
146         if (p.x > maxx){
147             maxx = p.x;
148         }
149         if (p.y < miny){
150             miny = p.y;
151         }
152         if (p.y > maxy){
153             maxy = p.y;
154         }
155     }
156     return new Point2D((minx+maxx)/2,(miny+maxy)/2);
157 }
158
159 public void apply(Matrix2D t){
160     for (int i = 0; i < pts2d.size(); i++){
161         Point2D p = t.apply((Point2D)(pts2d.get(i)));
162         pts2d.set(i,p);
163     }
164 }
165
166
167 public String toString() {
168     StringBuffer result = new StringBuffer(super.toString());
169     result.append(" " + pts2d.size());
170     for (int i = 0; i < pts2d.size(); i++) {
171         Point2D p = (Point2D)(pts2d.get(i));
172         result.append(" " + p);
173     }
174     return result.toString();
175 }
176
177 // Parse the coordinates
178 public void fromTokens(StringTokenizer st) throws MalformedURLException {
179     try {
180         selection = -1;
181         int n = Integer.parseInt(st.nextToken());
182         for (int i = 0; i < n; i++) {
183             pts2d.add(new Point2D(Double.parseDouble(st.nextToken()),
184                                 Double.parseDouble(st.nextToken())));
185         }
186     }
187 }
188 catch (NoSuchElementException e) {
189     throw new MalformedURLException(e.getMessage());
190 } catch (NumberFormatException e) {
191     throw new MalformedURLException(e.getMessage());
192 }
193 }
194
195 }

```

## FPEDShape.java

```

1  import java.awt.*;
2  import java.util.*;
3  import java.io.*;
4  import net.java.games.jogl.*;
5
6  /** This class represents a shape in the editor (such as a polygon)
7  @author Tim Lambert
8  */
9
10 public abstract class FPEDShape implements Cloneable,Serializable {
11     /** fill colour to fill the shape with */
12     Color fill = Color.white;
13

```

```

14     /** Texture for this shape */
15     MyTexture texture = null;
16
17     /** Texture scale for this shape */
18     double scale = 1.0;
19
20
21     /** Set the fill colour.
22      * @param fill colour to fill the shape with
23      */
24
25     public void setFillColor(Color fill){
26         this.fill= fill;
27     }
28
29     /** Get the fill colour.
30      * @returns fill colour to fill the shape with
31      */
32
33     public Color getFillColor(){
34         return fill;
35     }
36
37     /** Set the texture
38      * @param texture for this shape
39      */
40
41     public void setTexture(MyTexture texture){
42         this.texture = texture;
43     }
44
45
46     /** Get the texture
47      * @returns texture for this shape
48      */
49
50     public MyTexture getTexture(){
51         return texture;
52     }
53
54
55
56
57     /** Set the texture scale
58      * @param texture for this shape
59      */
60
61     public void setScale(double scale){
62         this.scale = scale;
63     }
64
65     /** get the texture scale
66      * @returns the scale
67      */
68
69     public double getScale(){
70         return scale;
71     }
72
73
74     /** This sets the current GL drawing colour
75      * @param gl The GL pipeline to use
76      * @param c colour to set GL to use
77      */
78     final static int SIZE = 2;
79     public void drawPoint(GL gl, double x, double y){
80         gl.glColor3f(1.0f, 1.0f, 1.0f);
81         gl.glPushAttrib(GL.GL_COLOR_BUFFER_BIT); //save current Logic Op
82         gl.glEnable(GL.GL_COLOR_LOGIC_OP);
83         gl.glLogicOp(GL.GL_XOR);
84         gl.glBegin(GL.GL_POLYGON);

```

```

85     gl.glVertex2d(x+SIZE,y+SIZE);
86     gl.glVertex2d(x-SIZE,y+SIZE);
87     gl.glVertex2d(x-SIZE,y-SIZE);
88     gl.glVertex2d(x+SIZE,y-SIZE);
89     gl.glEnd();
90     gl.glPopAttrib();
91 }
92
93 /** This sets the current GL drawing colour
94  * @param gl The GL pipeline to use
95  * @param c colour to set GL to use
96  */
97 public void setColor(GL gl, Color c){
98     gl.glColor3ub((byte)c.getRed(),(byte)c.getGreen(),(byte)c.getBlue());
99 }
100 /** Creates a shape from a string describing it.
101  * @param s The string describing the shape. This must consist of a sequence of
102  *         space separated fields. The first field is the name of a class
103  *         that is a child of FPEDShape. The next three are the colour.
104  *         The next is the name of the texture. Further fields will be interpreted
105  *         by the child class's fromTokens method.
106  * @return The clickable region described by the string.
107  * @exception MalformedShapeException If the string s cannot be parsed.
108  */
109
110 public static FPEDShape create(String s) throws MalformedShapeException{
111     FPEDShape shape;
112     String classname="";
113     try {
114         StringTokenizer st = new StringTokenizer(s, " \t");
115         classname = st.nextToken();
116         Class aclass = Class.forName(classname);
117         shape = (FPEDShape) aclass.newInstance();
118         shape.fill = new Color(Integer.parseInt(st.nextToken()),
119                               Integer.parseInt(st.nextToken()),
120                               Integer.parseInt(st.nextToken()));
121         shape.texture = (MyTexture) TextureChoice.textures.get(st.nextToken());
122         shape.scale = Double.parseDouble(st.nextToken());
123         shape.fromTokens(st);
124         return shape;
125     }
126     catch (ClassNotFoundException e) {
127         System.err.println("Class not found: "+e.getMessage());
128         throw new MalformedShapeException(s);
129     } catch (InstantiationException e) {
130         System.err.println("Couldn't Instance "+e.getMessage());
131         throw new MalformedShapeException(s);
132     } catch (IllegalAccessException e) {
133         System.err.println("Illegal Access "+e.getMessage());
134         throw new MalformedShapeException(s);
135     } catch (NoSuchElementException e) {
136         System.err.println("No such element "+e.getMessage());
137         throw new MalformedShapeException(s);
138     } catch (ClassCastException e) {
139         System.err.println(classname + "is not a subclass of FPEDShape");
140         throw new MalformedShapeException(s);
141     }
142 }
143
144 /** This must be implemented by child classes to parse the remaining fields
145  * of the string argument to create.
146  * @see FPEDShape#create
147  * @param context A URL. Relative URLs will be interpreted relative to this.
148  * @param st A StringTokenizer providing the remainig fields of the String argument to create.
149  * @exception MalformedShapeException If the string s cannot be parsed.
150  */
151
152 protected abstract void fromTokens(StringTokenizer st) throws MalformedShapeException;
153
154 /** This paints the shape on the screen.
155  * @param gl The GL pipeline to use

```

```

156     @param glu The GLU pipeline to use
157 */
158
159 public abstract void paint(GL gl, GLU glu, GLDrawable glc);
160
161 /** This paints a selected shape on the screen.
162     If a shape is selected first paint is called, then paintSelect.
163     @param gl The GL pipeline to use
164     @param glu The GLU pipeline to use
165 */
166
167 public abstract void paintSelect(GL gl, GLU glu);
168
169
170
171 /** This determines whether a query point is inside the shape.
172     @param p The query point
173     @returns true if the query point is inside the shape.
174 */
175
176 public abstract boolean contains(Point2D p);
177
178
179
180 /** This method translates the Shape by the specified amount.
181     @param deltax The number of pixels to change the x position by.
182     @param deltax The number of pixels to change the y position by.
183 */
184
185 public abstract void translate(double deltax, double deltax);
186
187 /** This method rotates the Shape by the specified amount.
188     @param fixed Centre of rotation
189     @param angle Angle to rotate by
190 */
191
192 public abstract void rotate(Point2D fixed, double angle);
193
194 /** This method returns the centre of a shape
195     @returns centre of rotation
196 */
197
198 public abstract Point2D centre();
199
200
201 /** This method adds a control point to an Shape.
202     What the control points mean depends on the shape. For a polygon the
203     control points are the corners. For a rectangle, they are two diagonally
204     opposite corners and so on for other sorts of shapes.
205     <p>This method can be used to interactively create an FPEDShape.
206     @param p control point
207 */
208
209 public abstract void addPoint(Point2D p);
210
211
212 /** This method selects the control point to be modified by the
213     setPoint method. This would typically be the closest one to that specified
214     in the parameters.
215     <p>This method can be used in conjunction with setPoint to interactively
216     modify a FPEDShape.
217     @see FPEDShape#setPoint
218     @param p control point
219 */
220
221 public abstract Point selectPoint(Point2D p);
222
223
224 /** This method gives the selected control point a new value. The
225     selected control point is the one specified by a previous selectPoint,
226     or the one must recently added.

```

```

227     @param p control point
228     */
229
230     public abstract void setPoint(Point2D p);
231
232     /** This method removes the selected control point. The
233         selected control point is the one specified by a previous selectPoint,
234         or the one must recently added.
235         @param p control point
236         */
237
238     public abstract void removePoint();
239
240
241     /** The string that this returns must be suitable to be used in create to
242         recreate the instance.
243         @returns The string description of this instance.
244         */
245     public String toString() {
246         return getClass().getName() + " " + fill.getRed() + " " +
247             fill.getGreen() + " " + fill.getBlue() + " " + texture +
248             " " + scale + " ";
249     }
250
251     public Object clone(){
252
253         try {
254             return create(toString());
255         } catch (MalformedShapeException e){
256             System.err.println("Could not clone FPEDShape "+e.getMessage());
257             return null;
258         }
259     }
260
261 }

```

## Matrix2D.java

```

1     /** This class implents 2D affine transformation matrices
2     */
3     import java.awt.*;
4
5     public class Matrix2D {
6         // the last row is always 0 0 1, so just need to specify six numbers
7         // giving the first two rows.
8
9         private double xx,xy,xw;
10        private double yx,yy,yw;
11        // last row is 0, 0, 1
12        public Matrix2D(double xx, double xy, double xw,
13            double yx, double yy, double yw) {
14            this.xx = xx;
15            this.xy = xy;
16            this.xw = xw;
17            this.yx = yx;
18            this.yy = yy;
19            this.yw = yw;
20        }
21
22        public Matrix2D() {
23            this(0,0,0,0,0,0);
24        }
25
26        // composition of two transformations - the resulting of first doing this
27        // one and then the one supplied as a parameter.
28
29        public Matrix2D compose(Matrix2D m) {
30            Matrix2D result = new Matrix2D();
31            result.xx = m.xx*xx + m.xy*yx;
32            result.xy = m.xx*xy + m.xy*yy;

```

```

33     result.xw = m.xx*xw + m.xy*yw + m.xw;
34     result.yx = m.yx*xx + m.yy*yx;
35     result.yy = m.yx*xy + m.yy*yy;
36     result.yw = m.yx*xw + m.yy*yw + m.yw;
37     return result;
38 }
39
40 // apply a transformation to a point
41
42 public Point apply(Point p) {
43     return new Point((int)Math.round(xx*p.x + xy*p.y + xw),
44                     (int)Math.round(yx*p.x + yy*p.y + yw));
45 }
46
47 public Point2D apply(Point2D p) {
48     return new Point2D(xx*p.x + xy*p.y + xw,
49                       yx*p.x + yy*p.y + yw);
50 }
51
52
53 //return rotation about point
54 public static Matrix2D rotateAbout(Point2D c,double angle){
55     return (new Matrix2D(1,0,-c.x,0,1,-c.y)).
56           compose(new Matrix2D(Math.cos(angle),-Math.sin(angle),0,
57                               Math.sin(angle),Math.cos(angle),0)).
58           compose(new Matrix2D(1,0,c.x,0,1,c.y));
59 }
60
61 //return scale about point
62 public static Matrix2D scaleAbout(Point2D c,double sx,double sy){
63     return (new Matrix2D(1,0,-c.x,0,1,-c.y)).
64           compose(new Matrix2D(sx,0,0,
65                               0,sy,0)).
66           compose(new Matrix2D(1,0,c.x,0,1,c.y));
67 }
68
69
70 // useful for debugging
71
72 public String toString() {
73     return xx + " " + xy + " " + xw + " | " +
74           yx + " " + yy + " " + yw;
75 }
76
77 }

```

## Point2D.java

```

1  /** This class represents a 2D point. Like Point of java.awt, but
2   * uses floats instead of ints.
3   */
4  import java.awt.Point;
5  import java.io.*;
6
7  public class Point2D implements Serializable{
8      double x,y;
9
10     public Point2D(double x, double y) {
11         this.x = x;
12         this.y = y;
13     }
14
15     public Point2D(Point p) {
16         this(p.x,p.y);
17     }
18
19     public double getX(){
20         return x;
21     }
22

```

```

23     public double getY(){
24         return y;
25     }
26
27     //interpolate this + t*(q-this)
28     public Point2D interp(Point2D q,double t){
29         return new Point2D(x + t*(q.x-x), y + t*(q.y-y));
30     }
31
32     public String toString(){
33         return x+" "+y;
34     }
35
36
37     public Point toPoint(){
38         return new Point((int)Math.round(x),(int)Math.round(y));
39     }
40 }

```

## PolygonTool.java

```

1     import java.awt.*;
2     import java.awt.event.*;
3     import java.util.*;
4
5     /** Polygon tool for graphical editor
6     @author Tim Lambert
7     */
8
9     public class PolygonTool extends Tool{
10
11
12         public PolygonTool(FPEDEdit editor) {
13             super(editor);
14         }
15
16         public String getName() {
17             return "Polygon";
18         }
19
20         public String getMessage() {
21             return "Click at each corner, Shift Click on last point";
22         }
23
24         protected FPEDPolygon p; /* Polygon being created */
25
26         // create the polygon
27         public void mouseClicked(MouseEvent e) {
28             if (p==null){
29                 p = new FPEDPolygon();
30                 editor.getProperties(p);
31
32                 canvas.getShapes().add(p);
33                 canvas.select(p);
34                 p.addPoint(gridPoint(e));
35             }
36             if (e.isShiftDown()){
37                 p = null;
38             } else {
39                 p.addPoint(gridPoint(e));
40             }
41             canvas.repaint();
42         }
43
44         public void mouseMoved(MouseEvent e) {
45             mouseDragged(e);
46         }
47
48         public void mouseDragged(MouseEvent e) {
49             if (p != null){

```

```

50         p.setPoint(gridPoint(e));
51         canvas.repaint();
52     }
53 }
54
55
56 }

```

## Tool.java

```

1  import java.awt.*;
2  import java.awt.event.*;
3
4  /** This class represents a tool such as select, translate etc in
5      the Bezier editor.
6  @author Tim Lambert
7      */
8
9  public abstract class Tool implements MouseMotionListener, MouseListener{
10
11
12     /** The canvas that the tool operates on*/
13     protected FPEDCanvas canvas;
14     /** The frame that contains the the canvas */
15     protected FPEDEdit editor;
16
17     /** The constructor for a Tool
18         @param editor - the frame that the tool is operating in.
19     */
20
21     public Tool(FPEDEdit editor) {
22         this.editor = editor;
23         canvas = editor.getCanvas();
24     }
25
26     /** Return the name of the tool to be displayed in a menu
27         @returns the name of the tool
28     */
29     public abstract String getName();
30
31
32     /** Return cursor to use when this tool is active
33         @returns the cursor to use when this tool is active
34     */
35
36     public Cursor getCursor() {
37         return new Cursor(Cursor.DEFAULT_CURSOR);
38     }
39
40     /** Returns the message to be displayed in the status area when the tool
41         is activated. The message will typically contain brief instructions.
42         @returns the message to be displayed
43     */
44
45     public String getMessage() {
46         return "";
47     }
48
49     /** mouse position, rounded to grid point */
50     Point2D gridPoint(MouseEvent e) {
51         int gridsize = editor.getGridSize();
52
53         return (new Point2D(((e.getX()+canvas.getXoffset())/gridsize)*gridsize,
54                             ((e.getY()+canvas.getYoffset())/gridsize)*gridsize));
55     }
56
57     /** mouse position
58         We have to adjust the (x,y) coordinates returned in the mouse event
59         too allow for the position of the scrollbars */
60     Point2D getPoint(MouseEvent e) {

```

```

61         return (new Point2D(e.getX()+canvas.getXoffset(),
62                             e.getY()+canvas.getYoffset()));
63     }
64
65
66     public void mousePressed(MouseEvent e) {
67     }
68
69     public void mouseClicked(MouseEvent e) {
70     }
71
72     public void mouseReleased(MouseEvent e) {
73     }
74
75     public void mouseEntered(MouseEvent e) {
76     }
77
78     public void mouseExited(MouseEvent e) {
79     }
80
81     public void mouseDragged(MouseEvent e) {
82     }
83
84     public void mouseMoved(MouseEvent e) {
85     }
86
87
88 }

```

## TranslateTool.java

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.*;
4
5  /** Translate tool for image map editor
6  @author Tim Lambert
7  */
8
9  public class TranslateTool extends Tool{
10
11     public TranslateTool(FPEDEdit editor) {
12         super(editor);
13     }
14
15     public String getName() {
16         return "Translate";
17     }
18
19     public String getMessage() {
20         return "Press mouse button inside an object and drag it to a new position";
21     }
22
23     private FPEDShape r;
24     private Point2D lastp;
25
26     // On button down, highlight the region, and display a message
27     public void mousePressed(MouseEvent e) {
28         r = canvas.getShapes().findShape(getPoint(e));
29         if (r == null) return ;
30         canvas.select(r);
31         canvas.repaint();
32         lastp = gridPoint(e);
33     }
34
35     public void mouseDragged(MouseEvent e) {
36         if (r != null) {
37             Point2D newp = gridPoint(e);
38             r.translate(newp.x-lastp.x, newp.y-lastp.y);
39             lastp = newp;

```

```

40         canvas.repaint();
41     }
42 }
43
44 public Cursor getCursor() {
45     return new Cursor(Cursor.MOVE_CURSOR);
46 }
47
48
49
50 }

```

## Wall.java

```

1  /** This class represents a wall */
2  import java.awt.*;
3  import java.util.*;
4  import net.java.games.jogl.*;
5
6  public class Wall extends FPEDPolygon {
7
8      public float lineWidth() {
9          return 4.0f;
10     }
11
12     /** paint this curve into g.*/
13     public void paint(GL gl, GLU glu, GLDrawable glc){
14
15         if (fill != null){
16             setColor(gl,fill);
17             gl.glDisable( GL.GL_TEXTURE_2D );
18         }
19         gl.glLineWidth(lineWidth());
20         gl.glBegin( GL.GL_LINES ); //draw the wall
21         for(int i = 0; i < 2; i++) {
22             Point2D p = (Point2D)(pts2d.get(i));
23             gl.glVertex2d(p.x, p.y);
24         }
25         gl.glEnd();
26     }
27
28
29     /** add a control point */
30     public void addPoint(Point2D p) {
31         if (pts2d.size() < 2) {
32             super.addPoint(p);
33         }
34     }
35
36
37     /** remove specified control point */
38     public void removePoint() {
39         return;
40     }
41
42     public boolean contains(Point2D p){
43         Edge2D e = new Edge2D((Point2D)pts2d.get(0),
44                               (Point2D)pts2d.get(1));
45         Point2D pe = e.toLineSpace(p);
46         return (pe.getX() > 0 && pe.getX() < 1 && Math.abs(pe.getY()) < EPSILON);
47     }
48
49 }

```

## WallTool.java

```

1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.*;

```

```

4
5  /** Polygon tool for graphical editor
6  @author Tim Lambert
7      */
8
9  public class WallTool extends Tool{
10
11
12      public WallTool(FPEDEdit editor) {
13          super(editor);
14      }
15
16      public String getName() {
17          return "Wall";
18      }
19
20      public String getMessage() {
21          return "Click and drag to create a wall";
22      }
23
24      protected Wall p; /* Wall being created */
25
26
27      public Wall newWall(){
28          return new Wall();
29      }
30
31      // create the polygon
32      public void mousePressed(MouseEvent e) {
33          p = newWall();
34          editor.getProperties(p);
35
36          canvas.getShapes().add(p);
37          canvas.select(p);
38          p.addPoint(gridPoint(e));
39          p.addPoint(gridPoint(e));
40          canvas.repaint();
41      }
42
43
44      public void mouseDragged(MouseEvent e) {
45          if (p != null){
46              p.setPoint(gridPoint(e));
47              canvas.repaint();
48          }
49      }
50
51      public void mouseReleased(MouseEvent e) {
52          mouseDragged(e);
53          p = null;
54      }
55  }
56 }

```

## Local illumination equation

$$I = I_a k_a + \sum_{k=1 \dots n} f_{\text{att}}(d_k) I_k (k_d \bar{N} \cdot \bar{L}_k + k_s (\bar{V} \cdot \bar{R}_k)^n)$$

where

- $k_a$  = ambient reflection coefficient
- $k_d$  = diffuse reflection coefficient
- $k_s$  = specular reflection coefficient
- $\bar{N}$  = surface normal
- $I_a$  = Ambient light intensity
- $I_k$  = Intensity of light source  $k$
- $\bar{L}_k$  = Direction to light source  $k$
- $\bar{V}$  = Direction of viewer
- $\bar{R}_k = 2\bar{N}(\bar{N} \cdot \bar{L}_k) - \bar{L}_k$
- $n$  = Phong exponent
- $d_k$  = distance to light source  $k$
- $f_{\text{att}}()$  = light attenuation function

All the vectors ( $N, L_k, V$ ) should be normalized.

## Bezier curve

$$B(t) = \sum_{i=0 \dots 3} b_i(t) p_i$$

where

- $b_0(t) = (1-t)^3$
- $b_1(t) = 3t(1-t)^2$
- $b_2(t) = 3t^2(1-t)$
- $b_3(t) = t^3$
- $p_i$  = Control point  $i$

## B spline curve

$$B_j(t) = \sum_{i=0 \dots 3} b_i(t) p_{i+j}$$

where

- $b_0(t) = (-t^3 + 3t^2 - 3t + 1)/6$
- $b_1(t) = (3t^3 - 6t^2 + 4)/6$
- $b_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6$
- $b_3(t) = t^3/6$
- $p_i$  = Control point  $i$

## 2D Transformations

Translation ( $t_x, t_y$ )	Scaling ( $s_x, s_y$ )
$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Rotation by $\theta$	Window-Viewport
$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \frac{v_w}{w_w} & 0 & v_x - \frac{v_w}{w_w} w_x \\ 0 & -\frac{v_h}{w_h} & v_y + v_h + \frac{v_h}{w_h} w_y \\ 0 & 0 & 1 \end{bmatrix}$

## Vector operations

$$\begin{aligned}
 \hat{a} &= (a_x, a_y, a_z) \\
 \hat{a} \cdot \hat{b} &= a_x b_x + a_y b_y + a_z b_z \\
 \|\hat{a}\| &= \sqrt{a_x^2 + a_y^2 + a_z^2} \\
 \hat{a} \times \hat{b} &= (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)
 \end{aligned}$$

## 3D Transformations

$$\begin{array}{cc}
 T(t_x, t_y, t_z) & S(s_x, s_y, s_z) \\
 \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 \\
 R_X(\theta) & R_Y(\theta) \\
 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 \\
 R_Z(\theta) & \text{perspective} \\
 \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}
 \end{array}$$

## Viewing Transformation

$$\begin{aligned}
 \bar{n} &= \overline{VPN} / \|\overline{VPN}\| \\
 \bar{u} &= \overline{Vup} \times \overline{VPN} / \|\overline{Vup} \times \overline{VPN}\| \\
 \bar{v} &= \bar{n} \times \bar{u} \\
 M_{XYZ \rightarrow UVN} &= \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} T(-o_x, -o_y, -o_z)
 \end{aligned}$$

## Linear Interpolation

Parametric equation of line from  $\bar{a}$  to  $\bar{b}$  is

$$L(t) = \bar{a} + (\bar{b} - \bar{a})t, \quad 0 \leq t \leq 1$$

This is also the formula for linear interpolation.